



Abschlussprüfung Sommer 2021

Fachinformatiker für Anwendungsentwicklung
Dokumentation zur betrieblichen Projektarbeit

Plate-Stock-Planning

Webanwendung zur Steuerung von Druckplatten und Visualisierung
von Plattenspeichern

Abgabetermin: Dresden, den 10.05.2021

Prüfungsbewerber:

Elias Martin

<Anschrift>

<Ort>



FLYERALARM

made to impress

Ausbildungsbetrieb:

FLYERALARM Industrial Print GmbH

Zschoner Ring 9

01723 Kesselsdorf

Dieses Werk einschließlich seiner Teile ist **urheberrechtlich geschützt**. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Autors unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen sowie die Einspeicherung und Verarbeitung in elektronischen Systemen.

Inhaltsverzeichnis

Abbildungsverzeichnis	III
Tabellenverzeichnis	IV
Listings	V
Abkürzungsverzeichnis	VI
1 Einleitung	1
1.1 Projektumfeld	1
1.2 Projektziel	1
1.3 Projektbegründung	1
1.4 Projektschnittstellen	2
1.5 Projektabgrenzung	2
2 Projektplanung	2
2.1 Projektphasen	2
2.2 Abweichungen vom Projektantrag	3
2.3 Ressourcenplanung	3
2.4 Entwicklungsprozess	3
3 Analysephase	4
3.1 Ist-Analyse	4
3.2 Wirtschaftlichkeitsanalyse	4
3.2.1 „Make or Buy“-Entscheidung	4
3.2.2 Projektkosten	4
3.3 Soll-Analyse	5
4 Entwurfsphase	6
4.1 Zielplattform	6
4.2 Architekturdesign	6
4.3 Entwurf der Benutzeroberfläche	7
4.4 Datenmodell	7
4.5 Maßnahmen zur Qualitätssicherung	7
5 Implementierungsphase	8
5.1 Implementierung der Benutzeroberfläche	8
5.2 Implementierung der Geschäftslogik	10
6 Abnahmephase	12
7 Einführungsphase	12

8	Dokumentation	13
9	Fazit	13
9.1	Soll-/Ist-Vergleich	13
9.2	Lessons Learned	13
9.3	Ausblick	14
	Eidesstattliche Erklärung	15
A	Anhang	i
A.1	Detaillierte Zeitplanung	i
A.2	Oberflächenentwurf	ii
A.3	Screenshots der Anwendung	iii
A.4	Client-Funktion zum Zuweisen von Druckplatten zu einem Speicher	iv
A.5	Backend Endpunkt zum Zuweisen von Druckplatten zu einem Speicher	v
A.6	Backend Middleware zur Sicherung der Endpunkte	vi

Abbildungsverzeichnis

1	Modell für Logs	8
2	Anmeldemaske der Benutzeroberfläche	9
3	Hauptseite der Benutzeroberfläche mit Beispieldaten	9
4	Modal zum Zuweisen von Druckplatten zu einem Speicher	10
5	Erster Entwurf der Benutzeroberfläche	ii
6	Hauptseite der Benutzeroberfläche im Dunkelmodus	iii
7	Modal zum Zuweisen von Druckplatten im Dunkelmodus	iii

Tabellenverzeichnis

1	Zeitplanung	2
2	Kostenaufstellung	5
3	Soll-/Ist-Vergleich	13

Listings

1	Client-Funktion zum Zuweisen von Druckplatten	iv
2	Backend Endpunkt zum Zuweisen von Druckplatten zu einem Speicher	v
3	Backend Middleware zur Sicherung der Endpunkte	vi

Abkürzungsverzeichnis

API	Application Programming Interface
FA	Flyeralarm GmbH
FAIP	FLYERALARM Industrial Print GmbH
CI	Continuous Integration
CTP	Computer-to-Plate (Verfahren in der Druckvorstufe)

1 Einleitung

1.1 Projektumfeld

Die Flyeralarm GmbH (im Folgenden FA genannt) wurde 2002 gegründet und ist heute eine der führenden Online-Druckereien Europas mit Hauptsitz in Würzburg. Sie übernimmt dabei den Betrieb des Onlineshops, die Annahme und Abwicklung der Kundenaufträge und die Generierung von Druckbögen. Weitere Produktionsschritte, darunter hauptsächlich die Produktion selbst, sowie der Versand von Kundenaufträgen, werden von ihrer Tochtergesellschaft, der FLYERALARM Industrial Print GmbH (im Folgenden FAIP genannt), bewerkstelligt. Mit über 14 Jahren Erfahrung ist FAIP Marktführer im industriellen Druck in Europa. Dafür sind ca. 1200 Mitarbeiter an sechs Standorten in Deutschland im Einsatz.

Der Auftraggeber meines Projektes ist die FAIP. Für mein Projekt stellt mir diese diverse Werkzeuge und vorhandene Systeme zur Verfügung, darunter einen Desktop-Computer zum Programmieren, eine GitLab Instanz zur Quellcodeverwaltung und Continuous Integration (CI), einen Linux-Server mit installierter Docker-Software und ein Active Directory zur Authentifizierung von Nutzern.

1.2 Projektziel

Bei der Herstellung von Druckerzeugnissen im Computer-to-Plate (CTP) Verfahren finden Druckplatten, sowie bald auch ein Plattenlogistiksystem pro Standort für diese, ihren Einsatz. Solch ein Plattenlogistiksystem besitzt mehrere Speicher, in denen Druckplatten gelagert werden. Um die Kommunikation mit solchen Plattenlogistiksystemen zu ermöglichen, soll eine Webanwendung entwickelt werden.

Die entstehende Webanwendung soll dabei die Möglichkeit bieten, vorzugeben, welche Platten in welchem Speicher gelagert werden, indem sie von Mitarbeiter zugewiesen werden können, sowie die Übersicht der verfügbaren Plattenspeicher und deren Inhalt zu bewahren.

Dabei soll der Prozess des Zuweisens von Platten in eine Datenbank gespeichert werden, um dieses Vorgehen zukünftig nachverfolgen und eventuell teilweise automatisieren zu können.

1.3 Projektbegründung

Um das neue Logistiksystem in Betrieb zu nehmen, muss zwingend ein Weg zur Kommunikation mit diesem existieren. Da es aktuell noch keine Anwendung gibt, die diese Aufgabe erfüllt, muss eine entwickelt werden.

1.4 Projektschnittstellen

Die entstehende Anwendung muss an zwei weitere Systeme angebunden werden. Dabei handelt es sich um unser internes Rogler-System, aus dem Daten unserer Aufträge bezogen werden, sowie um ein System des Logistiksystem-Herstellers, welches für die Verplanung der Druckplatten und Verwaltung der Speicher notwendig ist. Beide Systeme verfügen über jeweils eine REST Schnittstelle (API), über welche die Kommunikation stattfindet.

Des weiteren muss eine Datenbank angebunden werden, in welcher der Verlauf der Buchungen / Verplanungen gespeichert wird. Dabei handelt es sich um eine MongoDB, welche firmenintern zur Verfügung steht.

1.5 Projektabgrenzung

Die REST Schnittstellen zur Kommunikation mit anderen Systemen stellen diese selbst bereit.

Die Datenbank zum Speichern des Verlaufs der Buchungen von Druckplatten muss nicht neu aufgesetzt, sondern nur um eine neue Collection erweitert werden.

Das Zielsystem, auf welchem die Anwendung unter Verwendung von Docker gehostet werden soll, steht bereits zur Verfügung.

Hardwareseitig wird alles bereitgestellt und ist nicht Teil des Projekts.

2 Projektplanung

2.1 Projektphasen

Das Projekt wird innerhalb von 70 Stunden entstehen und ist grob in die Phasen gemäß Tabelle 1 eingeteilt.

Projektphase	Geplante Zeit
Analysephase	9 h
Entwurfsphase	9 h
Implementierungsphase	30 h
Testphase	4 h
Einführungsphase	6 h
Erstellen der Dokumentation	8 h
Pufferzeit	4 h
Gesamt	70 h

Tabelle 1: Zeitplanung

Eine detailliertere Zeitplanung befindet sich im Anhang A.1: Detaillierte Zeitplanung auf Seite i.

2.2 Abweichungen vom Projektantrag

Im Vergleich zur ursprünglichen Version im Projektantrag wurde, nach weiteren Absprachen und Anpassungen der Vorgehensweise, die Zeitplanung überarbeitet. Dies war notwendig, um den optimierten Anforderungen gerecht zu werden. Zudem wurde eine Pufferzeit eingefügt, um besser auf spontane Änderungen reagieren zu können.

Die Anwenderdokumentation hat aktuell keine hohe Priorität. Die Gründe dafür finden sich in Abschnitt 8. Daher, sowie aus Zeitknappheit, wurde das Erstellen der Anwenderdokumentation aus der Zeitplanung entfernt.

2.3 Ressourcenplanung

Büroarbeitsplatz

- Desktop-PC mit Manjaro Linux und Entwicklertools (Projektdokumentation, Entwicklung, praktische Arbeiten, Informationsbeschaffung)
- Linux Server mit Docker (Deploy)

Heimarbeitsplatz

- Desktop-PC mit Arch Linux, Entwicklertools und Teams (Projektdokumentation, Entwicklung, Austausch mit Kollegen)

Mitwirkendes Personal

- Patrick Donath (Planung, Absprachen während der Entwicklung, Testen)
- Marco Thiergart (Projektabschluss, Testen)
- Mitarbeiter der CTP Abteilung (Feedback und Anpassungswünsche)

2.4 Entwicklungsprozess

Für mein Projekt habe ich den agilen Ansatz gewählt. Dieser bietet sich an, da das Endergebnis des Projektes noch nicht vollständig geplant werden kann, weil sich das neue Logistiksystem noch im Aufbau befindet. Die Anforderungen können sich so jeder Zeit ändern, was zudem häufiger vorkommen kann, da das Projekt unter kontinuierlichen Rücksprachen mit Mitarbeitern der CTP-Abteilung entsteht.

3 Analysephase

3.1 Ist-Analyse

Druckplatten werden nach ihrer Belichtung auf einem Plattenwagen gelagert und darauf später von einem Mitarbeiter händisch zur Druckmaschine gefahren. Da sich dieser Vorgang als zeitintensiv und unübersichtlich herausgestellt hat, wird dieser Prozess aktuell durch den Aufbau eines Plattenlogistiksystems automatisiert.

3.2 Wirtschaftlichkeitsanalyse

Die Umsetzung des Projekts wird sich nach schätzungsweise vier bis fünf Jahren lohnen (mehr Ersparnisse als Ausgaben). Der genaue Zeitpunkt lässt sich jedoch noch nicht bestimmen, da es das erste Projekt seiner Art ist.

Die zukünftig erwarteten Einsparungen belaufen sich auf folgende:

- Die Automation des neuen Systems soll es weniger fehleranfällig machen. Dadurch werden Kosten für falsch produzierte Produkte vermieden.
- Mitarbeiter müssen die Druckplatten nicht mehr händisch durch die Halle fahren und sparen somit Zeit, in der sie andere Arbeitsschritte verrichten können.

3.2.1 „Make or Buy“-Entscheidung

Da es sich hierbei um das weltweit erste Projekt seiner Art handelt, muss eine entsprechende Softwarelösung neu dafür entwickelt werden.

Das Projekt wird firmenintern entwickelt und nicht ausgelagert, da das die Kommunikation vereinfacht, flexibler bei spontanen Anpassungen ist und somit Zeit und Kosten spart.

3.2.2 Projektkosten

Die genauen Gesamtkosten (mit Aufbau des neuen Plattenlogistik-Systems) sind firmenintern vertraulich zu behandeln.

Betrachtet man nur den Teilbereich der Webanwendung des Projekt, welche ich im Rahmen meiner Projektarbeit erstellt habe, so lassen sich die Kosten wie folgt berechnen:

Beispielrechnung (verkürzt) Die Kosten für die Entwicklung der Webapp setzen sich sowohl aus Personal-, als auch aus Ressourcenkosten zusammen. Laut Tarifvertrag verdient ein Auszubildender im dritten Lehrjahr pro Monat 1000 € Brutto.

$$8 \text{ h/Tag} \cdot 220 \text{ Tage/Jahr} = 1760 \text{ h/Jahr} \quad (1)$$

$$1000 \text{ €/Monat} \cdot 12 \text{ Monate/Jahr} = 12000 \text{ €/Jahr} \quad (2)$$

$$\frac{12000 \text{ €/Jahr}}{1760 \text{ h/Jahr}} \approx 6,82 \text{ €/h} \quad (3)$$

Für die Nutzung von Ressourcen (Räumlichkeiten, Arbeitsplatz, etc.) wird ein pauschaler Stundensatz von 15 € angenommen. Für andere Mitarbeiter wird ein pauschaler Stundenlohn von 25 € angenommen.

Eine Aufstellung der Gesamtkosten befindet sich in Tabelle 2:

Vorgang	Zeit	Kosten pro Stunde	Kosten
Personalkosten Azubi	70 h	6,82 € + 15 € = 21,82 €	1527,40 €
Personalkosten Mitarbeiter	10 h	25 € + 15 € = 40 €	400,00 €
			1927,40 €

Tabelle 2: Kostenaufstellung

3.3 Soll-Analyse

Es gilt eine Webanwendung zu entwickeln, mit welcher das neue Plattenlogistiksystem gesteuert und überwacht werden kann.

Dazu zählt es, vorzugeben, welche Platten in welchem Speicher gelagert werden. Diese Zuweisung erfolgt durch Mitarbeiter der CTP-Abteilung.

Zur Vereinfachung werden innerhalb der Anwendung, sowie im Rogler System, mehrere zu einem Auftrag gehörende Platten zu Plattenbündeln zusammengefasst.

Die noch nicht einem Speicher zugewiesenen Plattenbündel sollen, sortiert nach deren Priorität, als anklickbare Cards dargestellt werden. Diese Cards sollen nach folgenden Attributen der Plattenbündel filterbar sein: ihrer ID, Auflage, Grammatik, Format, Farbigkeit und Priorität.

Per Klick auf eine solche Card soll die Zuweisung des Plattenbündels zu einem Plattenspeicher möglich sein, wobei vor der Zuweisung einzelne Platten des Bündels abwählbar sein müssen.

Der Prozess des Zuweisens von Platten soll in eine MongoDB geloggt werden.

Auch soll es möglich sein, die Übersicht der verfügbaren Plattenspeicher und deren Inhalt zu bewahren, indem die Webanwendung alle verfügbaren Speicher einschließlich Inhalt (Platten-IDs und Anzahl) und noch verfügbarem Platz anzeigt.

Die Nutzung der Webanwendung soll nur Mitarbeitern der CTP-Abteilung möglich sein. Um dies zu sichern, ist es notwendig, eine Authentifizierung mittels LDAP einzubauen. Nutzer der Webanwendung müssen sich dann vor der Nutzung mit ihren Windows-Anmeldedaten einloggen.

Um das Deployment nach Änderungen zu vereinfachen, soll die Webanwendung über eine Continuous Integration (CI) verfügen, welche durch unsere firmeninterne GitLab-Instanz ausgeführt wird. Diese soll automatisch nach jedem Commit ein neues Docker-Image bauen, dieses auf unseren Linux Docker Server kopieren und dort starten.

Nach vollständiger Umsetzung des Gesamtprojekts ergibt sich eine erhöhte Effizienz der Produktion.

4 Entwurfsphase

4.1 Zielplattform

Das Frontend der Webanwendung wird für die Nutzung in modernen Web-Browsern (Mozilla FireFox und Chromium) ausgelegt.

Das Backend soll mit NodeJS in einem Docker Container auf einem Linux Server lauffähig sein.

4.2 Architekturdesign

Zur Entwicklung meiner Webanwendung habe ich das Framework Next.js verwendet. Next kombiniert beide Teile einer Webanwendung, sodass Frontend und Backend in einer einzigen Codebase entwickelt werden können. Diese Vorgehensweise habe ich für mein Projekt gewählt, da sie einige Vorteile bietet:

- Erleichtertes Deployment
- Eine Programmiersprache für das gesamte Projekt (erleichtert das Einarbeiten, die Entwicklung und zukünftige Änderungen)
- Einsatz der selben Libraries in Frontend und Backend (vermeidet Inkompatibilitäten)
- Models und Utility-Funktionen müssen nur ein Mal erstellt werden

Zudem bietet Next noch weitere Vorteile, wie beispielsweise:

- Ist kompatibel mit den für das Projekt relevanten Zielplattformen
- TypeScript Unterstützung
- Dateisystem basiertes Routing
- Automatische Code- und Rendering-Optimierungen

Intern benutzt Next für das Frontend die Library React. Das ist in unserem Entwickler-Team willkommen, da wir bereits bestehende Projekte und Erfahrung mit React haben.

Als Programmiersprache habe ich mich für TypeScript entschieden. Diese basiert auf JavaScript, bietet im Vergleich zu purem JavaScript jedoch die Unterstützung von Datentypen, Interfaces und einigem mehr. Dies führt zu einer geringeren Fehleranfälligkeit und verständlicherem Code, was weiteren Mitarbeitern das Einarbeiten erleichtert, sowie für eine bessere Wartbarkeit sorgt.

Für den Datenaustausch zwischen Frontend und Backend, sowie Backend und den REST Schnittstellen der externen Systeme, wird das JSON Format benutzt. Dieses bietet sich an, da es von allen Schnittstellen und nativ von TypeScript unterstützt wird.

Die Logging-Daten werden ebenfalls im JSON Format in einer MongoDB (einer dokumentenorientierten Datenbank) gespeichert.

Um seitenübergreifende Daten im Frontend zu verwalten, wie beispielsweise Login-Tokens und Übersetzungen, habe ich mich entschlossen, Higher-Order-Components (Wrapper-Funktionen für Seiten und Komponenten) von React, in Verbindung mit dessen Context-API zu verwenden. Dieser Ansatz hat den Vorteil, dass keine weitere Library (wie z.B. das sonst sehr beliebte Redux) benötigt wird.

4.3 Entwurf der Benutzeroberfläche

Für die Benutzeroberfläche wird eine Weboberfläche verwendet. Diese hat den Vorteil, dass keine Client-Anwendung auf mehreren Rechnern installiert und regelmäßig aktualisiert werden muss.

Vor Beginn der Entwicklung wurde in Zusammenarbeit mit den Endanwendern ein graphischer Entwurf der Hauptseite erstellt. Dieser befindet sich im Anhang in Abbildung 5.

4.4 Datenmodell

Die Logging-Daten werden in einer MongoDB in dem Format gespeichert, welches in Abbildung 1 dargestellt ist.

Zu weiteren verwendeten Datenformaten sind keine Entwürfe notwendig, da diese durch Schnittstellen anderer Hersteller bereits vorgegeben sind.

4.5 Maßnahmen zur Qualitätssicherung

Die Qualitätssicherung der Anwendung wurde hauptsächlich durch die Anwender selbst, sowie durch mich durchgeführt. Um dies zu ermöglichen, wurde schon früh, noch während der Implementierung, das Deployment eingerichtet. Seither ist firmenintern für alle Beteiligten jederzeit eine aktuelle Version der Webanwendung zugänglich. Diese wurde benutzt, um schon während der Entwicklung kontinuierlich die aktuellen Funktionen und Entwürfe zu testen, um Fehler und Änderungswünsche an den

```
interface Ilog {  
  /** ID of what has happened */  
  event: 'plates_assigned' | 'test'  
  /** Whether the event was successful or not */  
  success: boolean  
  /** When the action has happened. Will be auto generated if not set */  
  date?: Date  
  /*  
  ---  
  All following fields are optional, depending on the event  
  ---  
  */  
  /** Name of who has triggered the event */  
  username?: string  
  /** 'jobID' of plate(s) affected by the event */  
  plateJobID?: string  
  /** Number of plate(s) affected by the event */  
  plateCount?: number  
  /** ID of the storage the plates are assigned to */  
  toStorageID?: number  
}
```

Abbildung 1: Modell für Logs

Anforderungen zusammenzutragen. Um die Tests zu vereinfachen, wurde die Datenbank hinter dem Plattenlogistik-System bei Bedarf temporär mit Testdaten befüllt oder geleert. Um mit den Daten aus dem Rogler-System zu testen, musste nichts weiter konfiguriert werden. Da die Anwendung auf das Rogler-System nur Leseberechtigungen hat, konnte problemlos mit Live-Daten getestet werden, ohne dabei die Produktion zu behindern.

Zu den Testkriterien gehören beispielsweise:

- Korrekte Darstellung der Benutzeroberfläche in verschiedenen Browsern
- Funktionierende Anmeldung mit Windows-Logindaten
- Korrekte und vollständige Darstellung der nicht zugewiesenen Platten aus Rogler
- Funktionalität der Filter
- Funktionalität des Zuweisens von Platten zu einem Speicher (und anschließendes Validieren dieser Zuweisung durch Kontrolle in der Datenbank des Plattenlogistik-Systems)

5 Implementierungsphase

5.1 Implementierung der Benutzeroberfläche

Die Benutzeroberfläche wurde mit HTML, CSS und TypeScript, unter Zuhilfenahme der in Next.js integrierten Library React erstellt. Dabei wurden einzelne Komponenten der Oberfläche, wie z.B. die Cards für die nicht zugewiesenen Platten und Speicher, oder ein Modal in eigene Dateien ausgelagert, um Code-Duplikate zu vermeiden, den Code übersichtlich zu halten und zukünftige Änderungen zu vereinfachen.

CSS Klassen, welche nur in einzelnen Komponenten benötigt werden, wurden mithilfe der Library "css-modules" eingebunden. Diese Library kümmert sich um die automatische Umbenennung von CSS

Klassen, um Namenskonflikte zwischen unterschiedlichen Klassen mit gleichem Namen zu vermeiden. Zudem lassen sich die so generierten Stylesheets besser von Next.js optimieren, sodass auf jeder Seite nur noch die aktuell benötigten Styles geladen werden müssen. Dies spart Netzwerkbandbreite und verkürzt die Ladezeit.

Die Farbauswahl der Oberfläche in der Standard-Einstellung richtet sich nach den offiziellen Unternehmensfarben. Zusätzlich wurde noch ein dunkles Farbschema eingebaut, um Mitarbeitern ein angenehmeres Arbeiten in der Nacht zu ermöglichen. Screenshots dazu befinden sich im Anhang A.3: Screenshots der Anwendung auf Seite iii (Abbildung 6 und Abbildung 7).

Öffnet ein Mitarbeiter die Webanwendung zum ersten Mal, so erscheint zuerst die Anmeldemaske (siehe Abbildung 2).

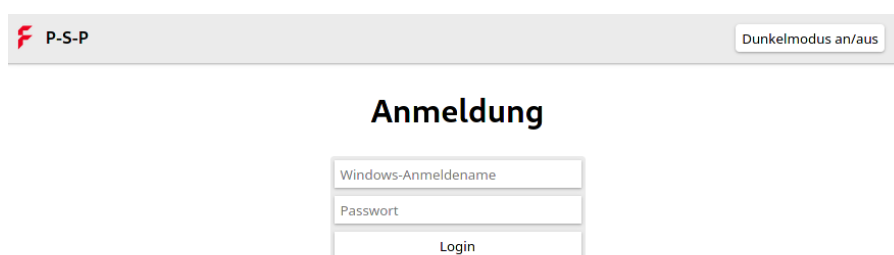
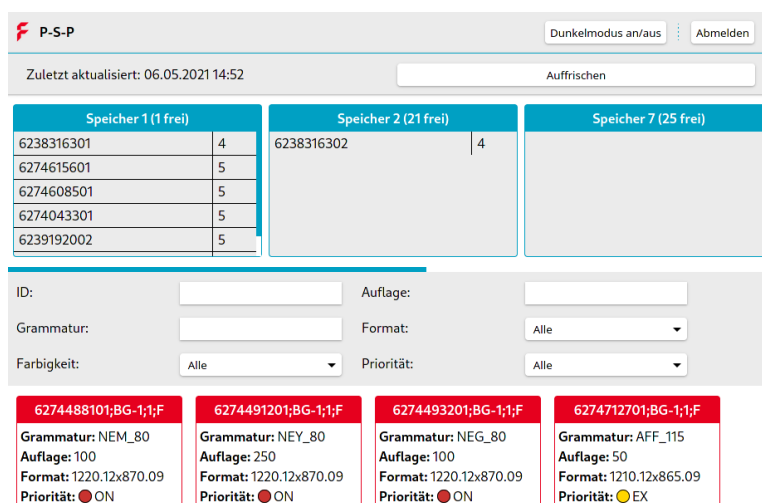


Abbildung 2: Anmeldemaske der Benutzeroberfläche

Dort ist es möglich, sich mit seinen Windows-Anmeldedaten einzuloggen. Sind die Daten korrekt, so erfolgt eine Weiterleitung zur Hauptseite der Anwendung (siehe Abbildung 3).



Speicher 1 (11 frei)		Speicher 2 (21 frei)		Speicher 7 (25 frei)	
6238316301	4	6238316302	4		
6274615601	5				
6274608501	5				
6274043301	5				
6239192002	5				

ID:	<input type="text"/>	Auflage:	<input type="text"/>
Grammatur:	<input type="text"/>	Format:	Alle
Farbigkeit:	Alle	Priorität:	Alle

6274488101;BG-1;1;F	6274491201;BG-1;1;F	6274493201;BG-1;1;F	6274712701;BG-1;1;F
Grammatur: NEM_80	Grammatur: NEY_80	Grammatur: NEG_80	Grammatur: AFF_115
Auflage: 100	Auflage: 250	Auflage: 100	Auflage: 50
Format: 1220.12x870.09	Format: 1220.12x870.09	Format: 1220.12x870.09	Format: 1210.12x865.09
Priorität: ON	Priorität: ON	Priorität: ON	Priorität: EX

Abbildung 3: Hauptseite der Benutzeroberfläche mit Beispieldaten

Im Vergleich zum ursprünglichen Entwurf wurden mehrere Anpassungen durchgeführt, z.B.:

- Es wurde ein zweiter Header ergänzt, um die Aktualität der angezeigten Daten darzustellen, sowie mit einem Button das neu Laden dieser zu ermöglichen.
- Die Anzeige der nicht zugewiesenen Plattenbündel wurde angepasst, um mehr Informationen über diese preiszugeben.

Klickt ein Nutzer auf ein nicht zugewiesenes Plattenbündel (Cards mit rotem Header, unten in Abbildung 3), so öffnet sich ein Modal (siehe Abbildung 4).

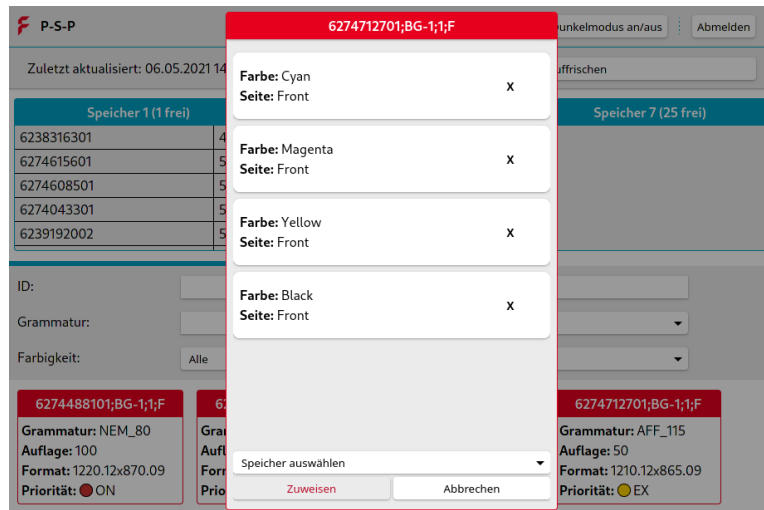


Abbildung 4: Modal zum Zuweisen von Druckplatten zu einem Speicher

In diesem Modal werden alle Druckplatten untereinander angezeigt, welche zum angeklickten Plattenbündel gehören. Einzelne Platten können hier mit einem Klick auf "X" abgewählt werden. Vor dem Klick auf "Zuweisen" kann der Nutzer in einem Dropdown noch den Speicher auswählen, auf welchem die Platten landen sollen. Die Speicher im Dropdown werden alle mit ihrer noch freien Kapazität aufgelistet. Wählt man einen Speicher, welcher keinen Platz mehr für die zuzuweisenden Platten hat, so deaktiviert sich der "Zuweisen" Button.

5.2 Implementierung der Geschäftslogik

Beispiel 1 Wird im Frontend (auf Seite des Clients) das Zuweisen von Platten ausgelöst, so wird die Funktion "handleAllocate" (Anhang A.4: Client-Funktion zum Zuweisen von Druckplatten zu einem Speicher auf Seite iv) ausgeführt.

Dieser wird als Parameter das Objekt "allocationData" mitgegeben. Dieses Objekt enthält zwei Variablen:

- pverr: ein Plattenbündel (enthält alle zuzuweisenden Platten)
- to: die ID des Speicher, auf den die Platten zugewiesen werden sollen

Es wird die Funktion "makeRequest" ausgeführt, wobei es sich um eine selbst erstellte Helfer-Funktion handelt, welche intern die library "fetch" benutzt. Bevor "makeRequest" den eigentlichen HTTP REST Request ausführt, prüft es z.B., ob der Benutzer schon angemeldet ist und sein Token noch gültig ist. Ist dies nicht der Fall, erfolgt eine automatische Weiterleitung zur Anmeldeseite.

Ist der Token gültig, so wird ein post Request an das Backend gesendet. Dieser enthält ein Array, in dem die zuzuweisenden Platten mitgegeben werden. Diese müssen jedoch vorher noch konvertiert werden, da sie im Rogler-System ein anderes Format haben, als sie vom Plattenlogistik-System angenommen werden. Diese Konvertierung übernimmt die Helfer-Funktion "convertRoglerToPspPlate".

Ist das Zuweisen abgeschlossen, so wird die Callback-Funktion in "then" ausgeführt. In dieser wird geprüft, ob es zu einem Fehler bei der Zuweisung kam. Ist dies der Fall, wird die Fehlermeldung "Platten konnten nicht zugewiesen werden!" angezeigt.

War die Zuweisung erfolgreich, so muss die React State Variable "storages" aktualisiert werden. Diese Variable enthält ein Array aller in der Benutzeroberfläche angezeigten Speicher, inklusive deren zugewiesenen Platten. Es wird nun in einer Schleife der Speicher gesucht, welchem die Platten zugewiesen wurden. Diese Platten werden nun auch auf das Array "assignedPlates" des Speichers hinzugefügt, sodass der aktualisierte Speicherinhalt in der Benutzeroberfläche wieder korrekt dargestellt wird.

Zudem wird die React State Variable "pverrs" aktualisiert. Diese enthält eine Auflistung aller noch nicht zugewiesenen Plattenbündel. Aus diesem Array wird nun das gerade zugewiesene Plattenbündel durch die Funktion "pverrs.filter" entfernt.

Beispiel 2 Das Backend stellt einen Endpunkt bereit, welcher für das Zuweisen von Druckplatten zu einem Speicher zuständig ist. Der Code dazu befindet sich in Anhang A.5: Backend Endpunkt zum Zuweisen von Druckplatten zu einem Speicher auf Seite v.

Dort ist ein TypeScript Modul zu sehen, dessen exportierte Funktion ("handler") automatisch von Next.js ausgeführt wird, sobald ein Request auf dem durch das Dateisystem vorgegebenen Pfad ankommt. Next.js übergibt an diese Funktion zwei Parameter: Ein Request und ein Response Objekt.

Die Funktion gibt einen Promise zurück, da sie asynchronen Code enthält. Ist die Funktion fertig, gibt diese die "resolve()" Funktion des Promise zurück. Dadurch weiß Next.js, dass es für den aktuellen Request nichts mehr zu tun gibt und beendet diesen intern.

Der Inhalt des Request-Body wird im JSON Format übertragen. Dieser enthält ein Array mit allen Druckplatten, welche zugewiesen werden sollen.

Vor der Zuweisung werden die durch den Client gesendeten Daten validiert (siehe die "if" Bedingungen, welche noch vor Aufruf der "makeRequest" Funktion geprüft werden). Wird dabei ein Problem erkannt, so sendet das Backend eine Antwort mit dem HTTP Status Code 400 (Bad Request). Dieser wird dann auf der Seite des Clients interpretiert und als Fehlermeldung dargestellt.

Sind die gesendeten Daten in Ordnung, so werden diese durch einen REST Request an die Schnittstelle des Plattenlogistik-Systems weitergeleitet. Dazu wird die Funktion "makeRequest" verwendet.

Das Ergebnis dieser Zuweisung wird dann durch die ebenfalls selbst erstellte Helfer-Funktion "dblog" in die Logging-Datenbank geschrieben.

Danach wird eine Antwort zurück an den Client gesendet, damit dieser weiß, ob die Zuweisung erfolgreich war (HTTP code 200) oder nicht (HTTP code 500).

Die zuweisende Funktion "handler" ist, wie in der letzten Zeile des Codebeispiels zu sehen, durch die Funktion "auth" umgeben. Der Code zu dieser befindet sich im nächsten Beispiel.

Beispiel 3 Im Backend ist eine Middleware-Funktion integriert, welche den Login-Status des Nutzers überprüft. Der Code dazu befindet sich in Anhang A.6: Backend Middleware zur Sicherung der Endpunkte auf Seite vi.

Dieser Funktion wird als Parameter eine Handler-Funktion übergeben, welche die eigentliche Logik des API-Endpunktes enthält.

Vor Ausführung dieser Handler-Funktion, wird der im Header "x-auth" enthaltene User-Token (JWT) überprüft. Ist dieser nicht gültig, so wird der HTTP status code 401 (Unauthorized) zurückgesendet. Nur wenn der Token gültig ist, wird die Handler-Funktion ausgeführt.

Um das lokale Testen zu vereinfachen, ist es möglich, durch das Setzen einer Umgebungsvariable auf "true", diese Überprüfung zu umgehen.

6 Abnahmephase

- Die Anwendung wurde von Marco Thiergart, sowie dem Abteilungsleiter der CTP-Abteilung abgenommen.
- Nach fertigem Aufbau des neuen Plattenlogistik-Systems wird sie auch allen restlichen Mitarbeitern der CTP-Abteilung präsentiert.
- Während der Abnahme wurden erneut alle Tests, wie sie in Unterabschnitt 4.5 beschrieben sind, erfolgreich durchgeführt.

7 Einführungsphase

- Die Anwendung wurde für die Ausführung in Docker konfiguriert.
- Es wurde die Continuous Integration (CI) in GitLab eingerichtet.
- Auf dem firmeninternen Proxy Server (nginx) wurde eine neue Subdomain konfiguriert, unter welcher die Webanwendung nun zu erreichen ist.

8 Dokumentation

Bei der Entwicklung der App wurde sehr auf intuitive Bedienbarkeit geachtet. Zudem entstand die Anwendung in enger Zusammenarbeit mit den Endanwendern, welche daher mit der Bedienung dieser bereits vertraut sind. Daher besteht aktuell keine Notwendigkeit, eine Benutzerdokumentation bereitzustellen.

Da einerseits IDEs, wie die bei dieser Entwicklung eingesetzte VSCode, heutzutage Tools zur Verfügung stellen, die es ermöglichen, sich schnell einen Überblick über die Anwendungsstruktur zu verschaffen und da andererseits während der Programmierung auf sauberen, gut strukturierten und, wo sinnvoll, kommentierten Code Wert gelegt wurde, wurde auch auf das Bereitstellen einer Entwicklerdokumentation verzichtet.

9 Fazit

9.1 Soll-/Ist-Vergleich

Das Projektziel der Webanwendung wurde zur Zufriedenheit der CTP-Abteilung erreicht.

Wie in Tabelle 3 zu erkennen ist, konnte die Zeitplanung dabei bis auf wenige Abweichungen dank der Pufferzeit eingehalten werden.

Phase	Geplant	Tatsächlich	Differenz
Analysephase	9 h	9 h	
Entwurfsphase	9 h	9 h	
Implementierungsphase	30 h	31 h	+1 h
Testphase	4 h	5 h	+1 h
Einführungsphase	6 h	7 h	+1 h
Erstellen der Dokumentation	8 h	9 h	+1 h
Pufferzeit	4 h	0 h	-4 h
Gesamt	70 h	70 h	

Tabelle 3: Soll-/Ist-Vergleich

9.2 Lessons Learned

- Ich habe gelernt, dass vor allem bei einem Projekt mit agiler Vorgehensweise eine größere Pufferzeit eingeplant werden muss.
- Aufgrund der straffen Zeitplanung habe ich gelernt, mir die Zeit besser einzuteilen und mich zuerst auf Kernfunktionen zu fokussieren.

- Next.js wurde mit dieser Anwendung das erste Mal in unserem Entwickler-Team verwendet, hat sich aber als optimal für Webanwendung dieser Art herausgestellt.

9.3 Ausblick

- In Zukunft sollen noch Übersetzungen in andere Sprache eingebaut werden.
- Es soll eine Unterseite eingebaut werden, auf welcher eine größere und detailliertere Übersicht der Speicher mit deren Inhalten dargestellt wird.
- Die Webanwendung wird sicherlich noch weitere Anpassungen benötigen. Welche genau das sind, wird sich nach fertigem Aufbau des neuen Plattenlogistik-Systems und während der Benutzung in der Produktion herausstellen.

Eidesstattliche Erklärung

Ich, Elias Martin, versichere hiermit, dass ich meine **Dokumentation zur betrieblichen Projektarbeit** mit dem Thema

Plate-Stock-Planning – Webanwendung zur Steuerung von Druckplatten und Visualisierung von Plattenspeichern

selbständig verfasst und keine nicht angegebenen Quellen und Hilfsmittel benutzt habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Dresden, den 10.05.2021

ELIAS MARTIN

A Anhang

A.1 Detaillierte Zeitplanung

Analysephase	9 h
1. Fachgespräch mit der CTP-Abteilung	3 h
2. „Make or buy“-Entscheidung und Wirtschaftlichkeitsanalyse	1 h
3. Herausfinden benötigter Libraries	2 h
4. Herausfinden benötigter Backend routes	1 h
5. Benötigte Schnittstellen von Fremdsystemen analysieren	2 h
Entwurfsphase	9 h
1. Benutzeroberflächen entwerfen	3 h
2. Funktionen der Backend routes planen	3 h
3. Datenbankentwurf	1 h
4. Struktur der Codebase erstellen	1 h
5. Docker Container entwerfen	1 h
Implementierungsphase	30 h
1. Entwicklung des Frontends	12 h
1.2 Umsetzung der HTML-Oberflächen und Stylesheets	4 h
1.3 React components programmieren	4 h
1.4 Implementation der Authentifizierung	2 h
1.5 Gerüst zum Ermöglichen von Übersetzungen einbauen	2 h
2. Entwicklung des Backends	10 h
2.1 Routes für Druckplattenmanagement erstellen	2 h
2.2 Requests an Schnittstellen programmieren	2 h
2.3 Implementation der Authentifizierung	3 h
2.3 Verbindung zur Datenbank programmieren	1 h
2.4 Logging implementieren	2 h
3. Utility Funktionen schreiben	2 h
4. Konfigurationsdateien erstellen	6 h
4.1 Next.js, ESLint, Prettier, VSCode	2 h
4.2 Docker	2 h
4.3 Continuous Integration	2 h
Testphase	4 h
1. Abnahmetest durch Fachabteilung	1 h
2. Code-Review durch Teamleiter	1 h
3. Tests während der Entwicklung	2 h
Einführungsphase	6 h
1. Aktivieren der Continuous Integration	2 h
2. Inbetriebnahme der Container auf dem Docker Server	2 h
3. Abnahme durch Teamleiter	1 h
4. Einführung und Erklärung für Mitarbeiter	1 h
Erstellen der Dokumentation	8 h
1. Erstellen der Projektdokumentation	8 h
Pufferzeit	4 h
1. Puffer	4 h
Gesamt	70 h

A.2 Oberflächenentwurf

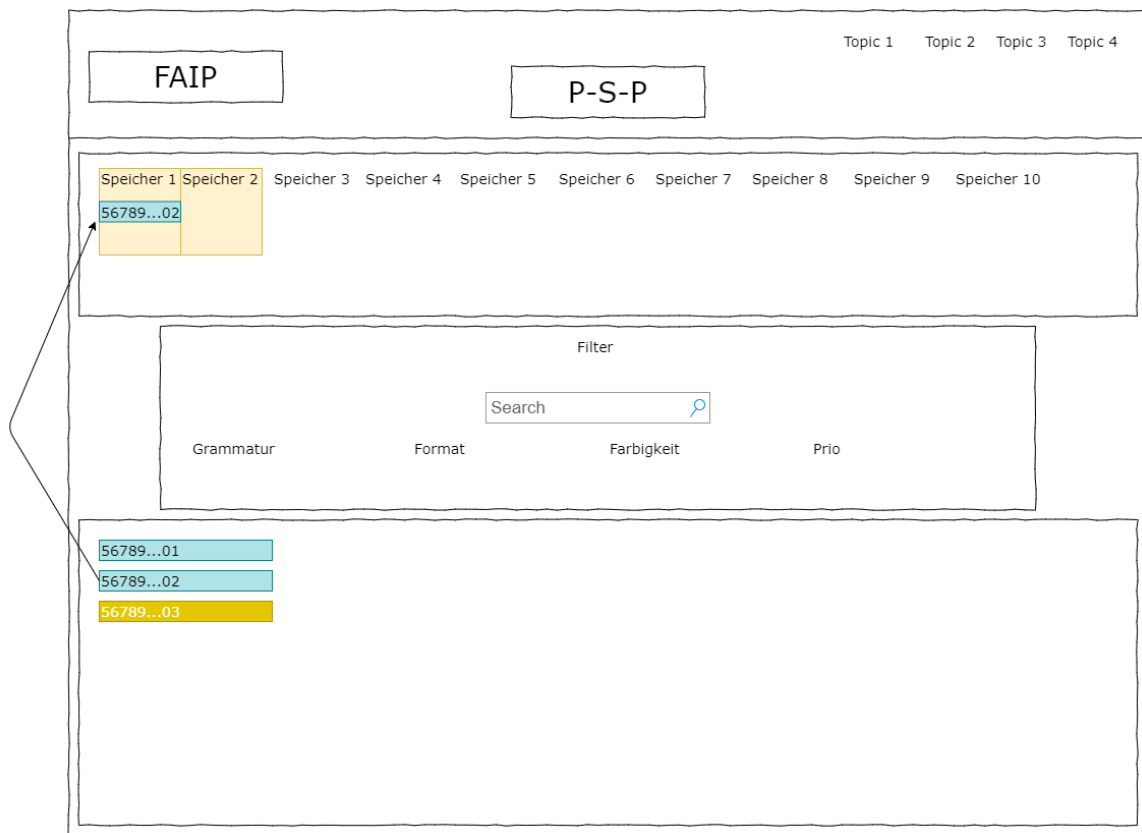


Abbildung 5: Erster Entwurf der Benutzeroberfläche

A.3 Screenshots der Anwendung

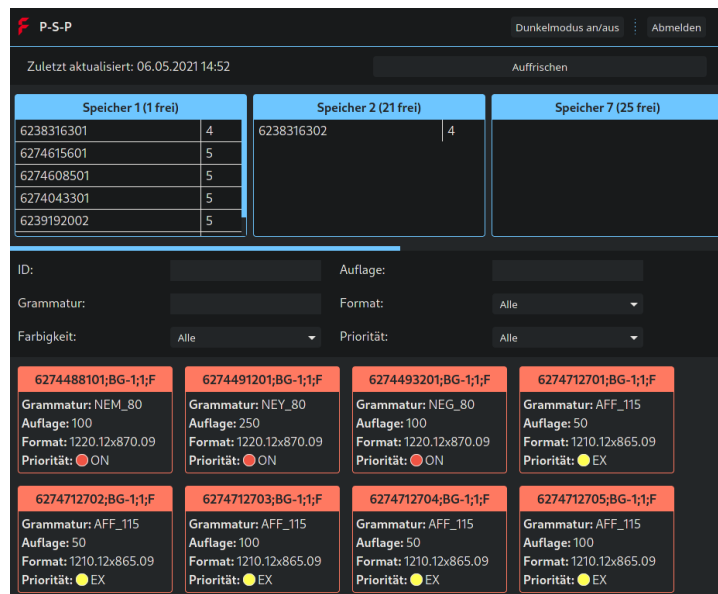


Abbildung 6: Hauptseite der Benutzeroberfläche im Dunkelmodus

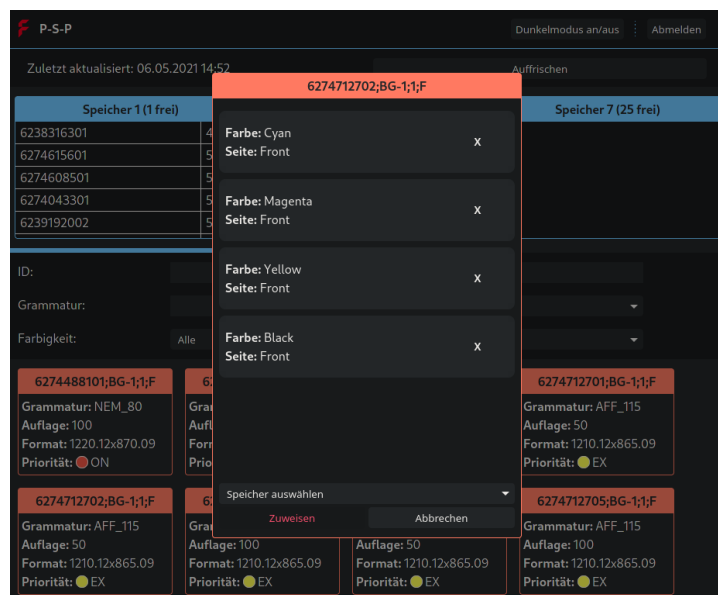


Abbildung 7: Modal zum Zuweisen von Druckplatten im Dunkelmodus

A.4 Client-Funktion zum Zuweisen von Druckplatten zu einem Speicher

```

1 const handleAllocate = (allocationData: IAllocationData) => {
2   makeRequest(
3     '/api/plates/assign-multi',
4     {
5       method: 'post',
6       headers: {
7         'Content-Type': 'application/json'
8       },
9       body: JSON.stringify(
10        allocationData.pverr.platten.map((plate) =>
11          convertRoglerToPspPlate(
12            plate,
13            allocationData.pverr,
14            allocationData.to
15          )
16        )
17      )
18    }
19  ).then((res) => {
20    if (res.error) {
21      console.log('Failed to assign plates:', res.error)
22      setDataState((prevState) => ({
23        ...prevState,
24        errorText: 'Platten konnten nicht zugewiesen werden!'
25      })))
26    } else {
27      for (let i = 0; i < storages.length; i++) {
28        if (storages[i].storageID === allocationData.to) {
29          allocationData.pverr.platten.forEach((plate) => {
30            const pspPlate = convertRoglerToPspPlate(plate)
31            storages[i].assignedPlates.unshift(pspPlate)
32          })
33          break
34        }
35      }
36      setStorages ([... storages])
37      setPverrs(
38        pverrs.filter (
39          (pverr) => pverr.plateID !== allocationData.pverr.plateID
40        )
41      )
42    }
43  })
44 }

```

Listing 1: Client-Funktion zum Zuweisen von Druckplatten

A.5 Backend Endpunkt zum Zuweisen von Druckplatten zu einem Speicher

```

1 import type { NextApiRequest, NextApiResponse } from 'next'
2 import { makeRequest } from 'lib/Utils/fetcher'
3 import auth from 'lib/middleware/auth'
4 import { IPspPlate } from 'types/Plate'
5 import { dblog } from 'lib/Utils/mongodb-logger'
6
7 function handler(req: NextApiRequest, res: NextApiResponse): Promise<void> {
8   return new Promise((resolve) => {
9     if (!req.body) {
10       res.status(400).send({
11         message: 'Request body cannot be empty'
12       })
13       return resolve()
14     }
15
16     if (req.headers['content-type'] !== 'application/json') {
17       res.status(400).send({
18         message: 'Invalid Content-Type header'
19       })
20       return resolve()
21     }
22
23     if (!Array.isArray(req.body)) {
24       res.status(400).send({
25         message: 'Body is not an Array'
26       })
27       return resolve()
28     }
29
30     const plates: IPspPlate[] = req.body
31
32     if (plates.length === 0) {
33       res.send('')
34       return resolve()
35     }
36
37     makeRequest(
38       `${process.env.LOGISTIKSYSTEM_API_URL}/Platten/Bulk`,
39       {
40         method: 'post',
41         headers: {
42           'Content-Type': 'application/json'
43         },
44         body: JSON.stringify(plates)
45       }
46     ).then((result) => {
47       if (result.error) {
48         dblog({
49           event: 'plates_assigned',
50           success: false,
51           plateJobID: plates[0].jobID,
52           plateCount: plates.length,
53           toStorageID: plates[0].storage
54         })
55         res.status(500).json({
56           message: 'Failed assign plates'
57         })
58       } else {
59         dblog({
60           event: 'plates_assigned',
61           success: true,
62           plateJobID: plates[0].jobID,
63           plateCount: plates.length,
64           toStorageID: plates[0].storage
65         })
66         res.send('')
67       }
68       return resolve()
69     })
70   })
71 }
72
73 export default auth(handler)

```

Listing 2: Backend Endpunkt zum Zuweisen von Druckplatten zu einem Speicher

A.6 Backend Middleware zur Sicherung der Endpunkte

```
1 import type { NextApiRequest, NextApiResponse } from 'next'
2 import jwt from 'jsonwebtoken'
3 import IUser from 'types/IUser'
4
5 export default function (
6   handler: (req: NextApiRequest, res: NextApiResponse) => void
7 ): (
8   req: NextApiRequest & { authMiddleware: { user: IUser } },
9   res: NextApiResponse
10 ) => void {
11   return (req, res) => {
12     if (process.env.NEXT_PUBLIC_DISABLE_AUTH === 'true') {
13       return handler(req, res)
14     }
15
16     const token = req.headers['x-auth']
17     try {
18       const decoded = jwt.verify(token, process.env.JWT_SECRET)
19       req.authMiddleware = { user: decoded.user }
20       return handler(req, res)
21     } catch (_e) {
22       return res.status(401).send('')
23     }
24   }
25 }
```

Listing 3: Backend Middleware zur Sicherung der Endpunkte